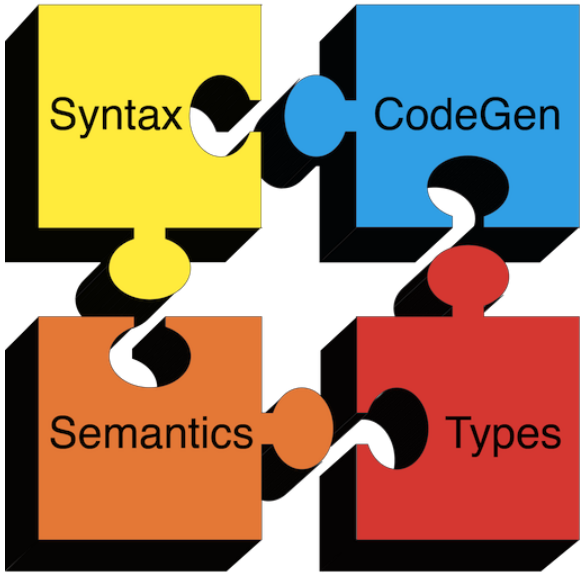
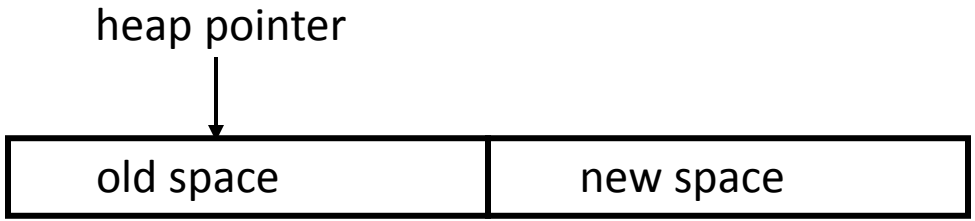


Compilers

Stop and Copy

Stop and Copy

- Memory is organized into two areas
  - old space: used for allocation
  - new space: used as a reserve for GC



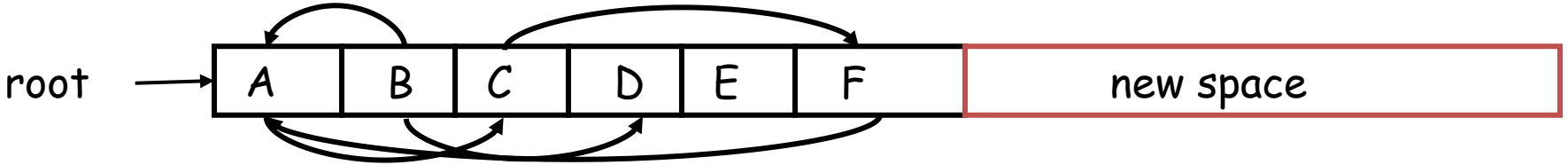
- The heap pointer points to the next free word in the old space
- Allocation just advances the heap pointer

Stop and Copy

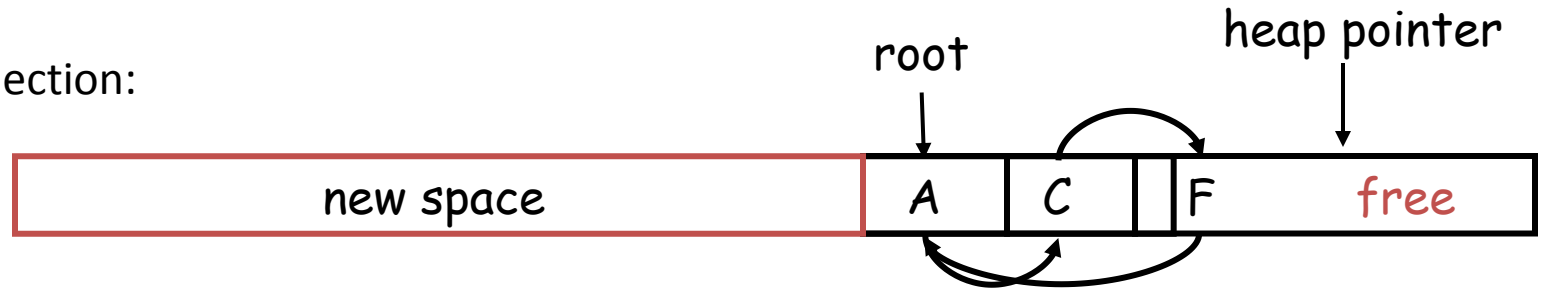
- Starts when the old space is full
- Copies all reachable objects from old space into new space
  - garbage is left behind
  - after the copy phase the new space uses less space than the old one before the collection
- After the copy the roles of the old and new spaces are reversed and the program resumes

Stop and Copy

Before collection:



After collection:

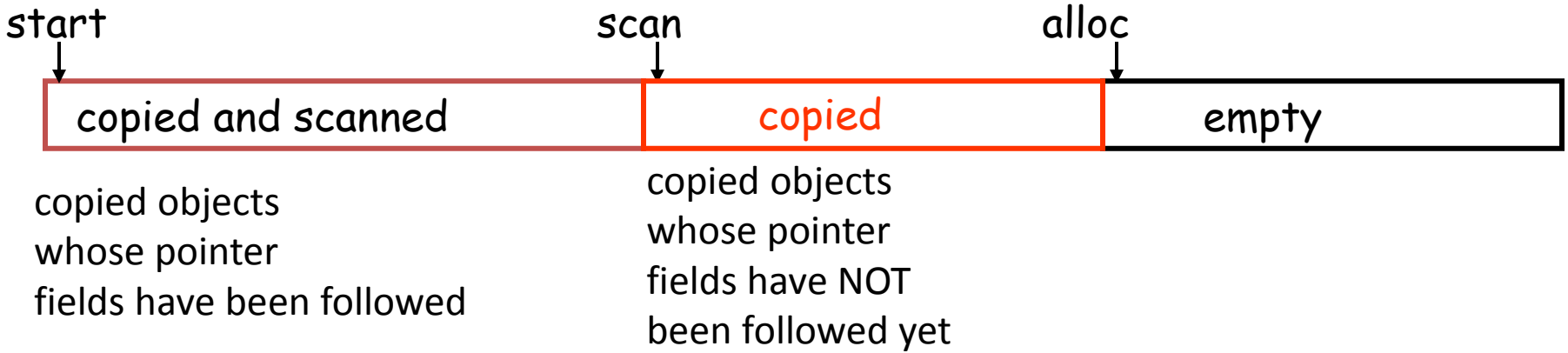


Stop and Copy

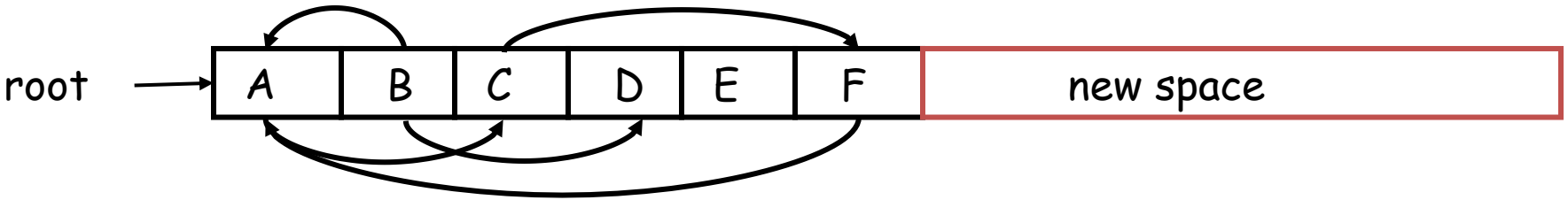
- We need to find all the reachable objects, as for mark and sweep
- As we find a reachable object we copy it into the new space
  - And we have to fix ALL pointers pointing to it!
- As we copy an object we store in the old copy a forwarding pointer to the new copy
  - when we later reach an object with a forwarding pointer we know it was already copied

Stop and Copy

- We still have the issue of how to implement the traversal without using extra space
- The following trick solves the problem:
  - partition the new space in three contiguous regions

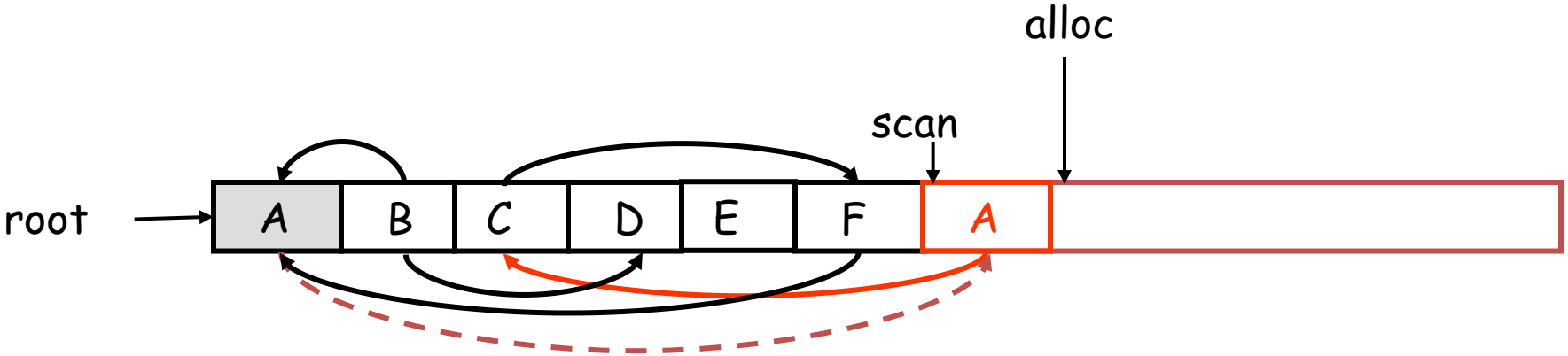


Stop and Copy



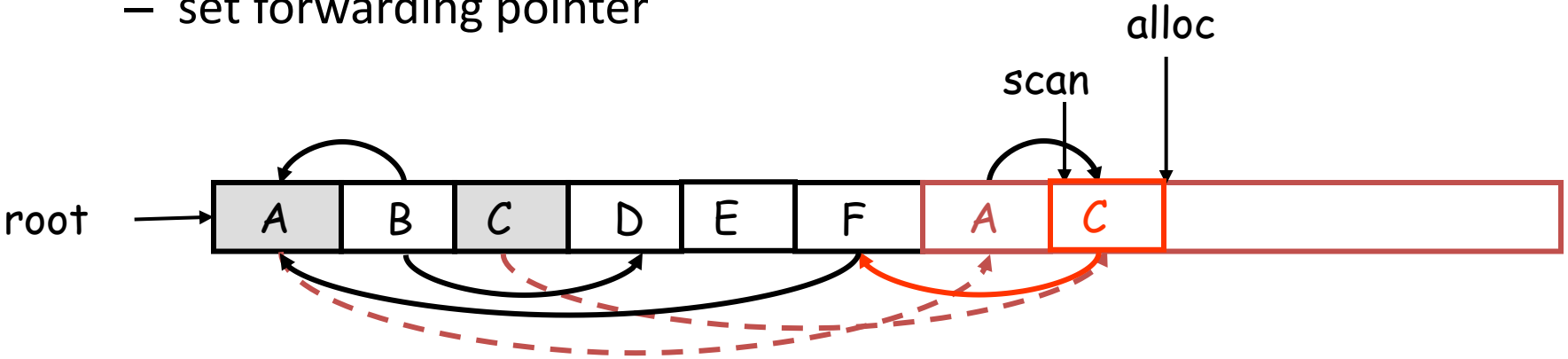
Stop and Copy

- Step 1: Copy the objects pointed to by roots and set forwarding pointers



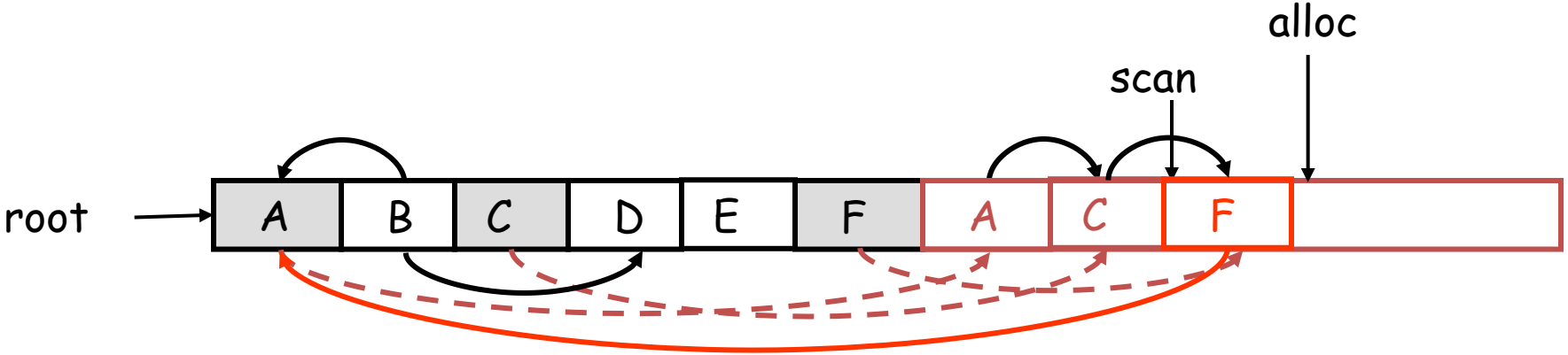
Stop and Copy

- Step 2: Follow the pointer in the next unscanned object (A)
  - copy the pointed-to objects (just C in this case)
  - fix the pointer in A
  - set forwarding pointer



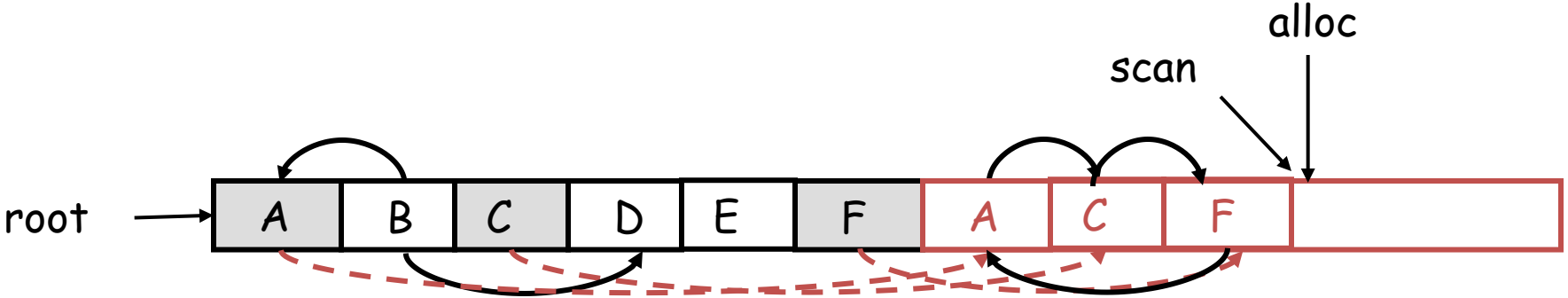
Stop and Copy

- Follow the pointer in the next unscanned object (C)
  - copy the pointed objects (F in this case)



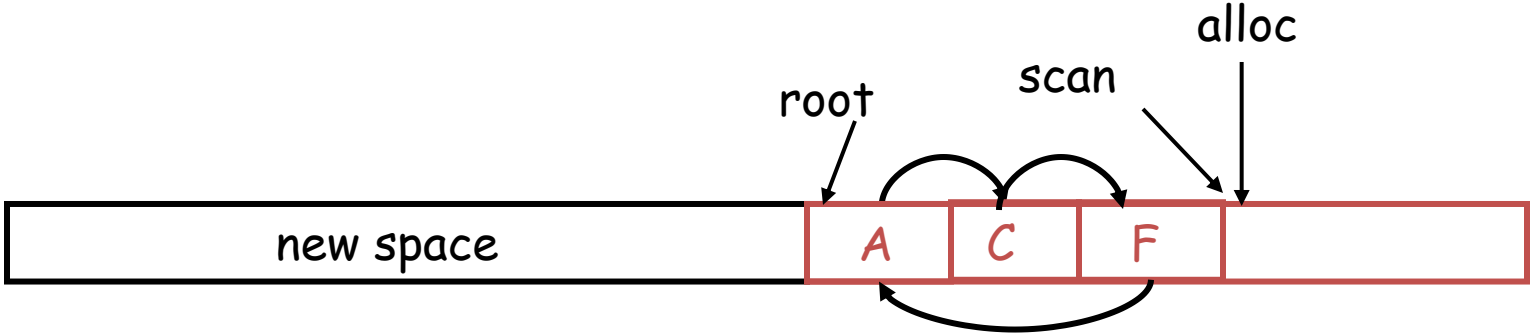
Stop and Copy

- Follow the pointer in the next unscanned object (F)
  - the pointed object (A) was already copied. Set the pointer same as the forwading pointer



Stop and Copy

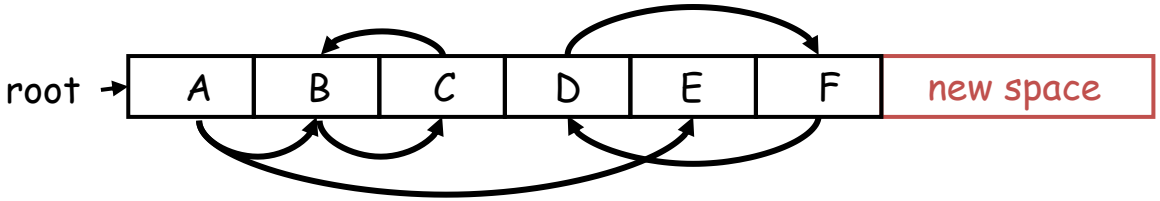
- Since scan caught up with alloc we are done
- Swap the role of the spaces and resume the program



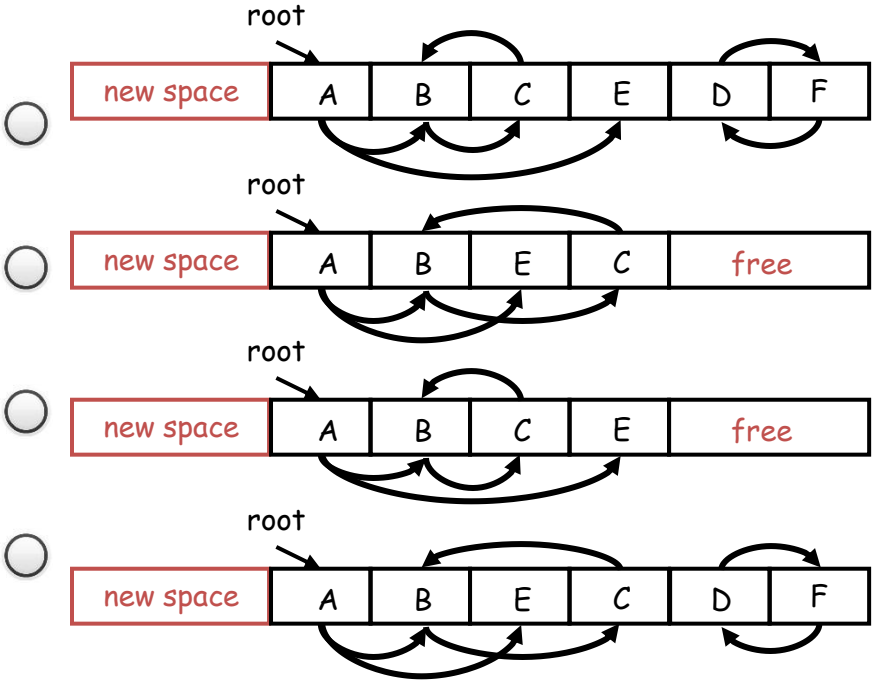
Stop and Copy

```
while scan <> alloc do
    let O be the object at scan pointer
    for each pointer p contained in O do
        find O' that p points to
        if O' is without a forwarding pointer
            copy O' to new space (update alloc pointer)
            set 1st word of old O' to point to the new copy
            change p to point to the new copy of O'
        else
            set p in O equal to the forwarding pointer
        fi
    end for
    increment scan pointer to the next object
od
```

Stop and Copy



Choose the correct final heap after stop and copy garbage collection.



Stop and Copy

- As with mark and sweep, we must be able to tell how large an object is when we scan it
  - and we must also know where the pointers are inside the object
- We must also copy any objects pointed to by the stack and update pointers in the stack
  - this can be an expensive operation

Stop and Copy

- Stop and copy is generally believed to be the fastest GC technique
- Allocation is very cheap
  - just increment the heap pointer
- Collection is relatively cheap
  - especially if there is a lot of garbage
  - only touch reachable objects
- But some languages do not allow copying
  - C, C++